MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Michal Vacek

Stolní hra Agricola a umělá inteligence

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.
Studijní program: Informatika
Studijní obor: Programování a softwarové systémy

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V ..... dne ....

Podpis

Chtěl bych poděkovat vedoucímu bakalářské práce RNDr. Tomáši Holanovi, Ph.D. za jeho úžasné nápady při návrzích, kam s prací směřovat, za skvělé připomínky a za všechen čas, který mi při tvorbě této práce věnoval.

Název práce: Stolní hra Agricola a umělá inteligence

Autor: Michal Vacek

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: Bakalářská práce na téma umělá inteligence ke hře Agricola se zabývá samotnou implementací hry, počítačem řízeného hráče a vytvořením strategie pro lidského hráče. Vyvinutý program umožňuje v základním rozsahu hru uživatele s počítačem, paralelní simulaci her počítačů proti sobě, analýzu dat a zobrazení historie hry.

Klíčová slova: umělá inteligence, stolní hra, Agricola, strategie hry

Title: Agricola the board game and artificial intelligence

Author: Michal Vacek

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D., Department of Software and Computer Science Education

Abstract: Bachelor thesis on artificial intelligence for Agricola deals with the actual implementation of the game, the computer controlled player, and the creation of a strategy for the human player. The developed program allows, in a basic scope, user-computer play, parallel simulation of computer games against each other, data analysis, display of game history.

Keywords: artificial intelligence, board game, Agricola, game strategy

Obsah

Úvod ... 3
1 Pravidla ... 5
1.1 Použité pojmy ... 5
1.2 Pravidla hry ... 6
2 Implementace hry ... 8
2.1 Struktura programu ... 8
2.2 Použití funkcionálních principů ... 8
2.3 Reprezentace dat a jejich ukládání ... 9
3 Umělá inteligence ... 11
3.1 Problematika dokonalé hry ... 11
3.2 Použité implementace ... 11
3.2.1 Náhodný hráč ... 12
3.2.2 Monte Carlo tree search ... 12
3.2.3 Minimax algoritmus ... 13
3.2.3.1 Alfa-Beta MinMax algoritmus ... 14
4 Výsledky hráčů ... 16
4.1 Porovnání mezi různými typy hráčů ... 16
4.2 Vyvození základních poznatků z výsledků ... 17
4.3 Shrnutí strategie ... 17
4.4 Implementace strategie ... 18
4.4.1 Výsledky strategie ... 19
5 Uživatelská dokumentace ... 21
5.1 Režim příkazového řádku ... 21
5.1.1 Příkaz Game ... 22
5.1.2 Příkaz Data ... 23
5.2 Režim grafického okna ... 24
5.2.1 Nová hra ... 24
5.2.1.1 Obrazovka s probíhající hrou ... 25
5.3 Načíst hru ... 27
5.3.1 Obrazovka s dokončenou hrou ... 28
5.4 Hra umělých inteligencí ... 28

6 Programátorská dokumentace ... 30
6.1. Game ... 31
6.2 GameAI ... 32
6.3 Extensions ... 32
6.4 Serialization ... 33
Závěr ... 34
Seznam tabulek ... 35
Seznam obrázků ... 36
Seznam použité literatury ... 37
A Přílohy ... 38
A.1 Elektronická příloha ... 38

Úvod

Agricola je desková hra pro 1 až 5 hráčů od autora Uwe Rosenberg a poprvé byla vydána v roce 2007.

Obrázek 1: Obsah deskové hry Agricola. (Zdroj https://mindok.cz/hra/agricola-zakladni-hra/)

Tématem této práce bylo za pomocí umělé inteligence vytvořit obecnou strategii a ověřit její funkčnost. K tomu bylo zapotřebí vytvořit reprezentaci této deskové hry v počítačové verzi s možností lehce vytvářet a přidávat nové umělé inteligence.

Tato hra má již existující počítačovou verzi na platformě Steam a je tedy možné si hru plně zahrát tam. Ovšem neexistuje žádná oficiální strategie a strategie popsané na internetu jsou velmi obecné a nejsou nijak ověřené. Jelikož primárním cílem nebylo vytvořit hru hratelnou člověkem, tak grafické rozhraní neimplementuje hru v plném rozsahu a jedná se jen o zjednodušenou verzi. Tyto limity ovšem neplatí pro umělou inteligenci.

Oproti Steam verzi nabízí tato verze jednoduché přidávání nových umělých inteligencí a jejich testování v rychlé hře díky možnosti odebrání grafického

prostředí. Dále umožňuje hru různých typů umělé inteligence proti sobě v paralelním běhu.

Práci jsem rozdělil do několika kapitol. První kapitolu jsem věnoval základním pravidlům hry a seznámení s používanými pojmy. V další kapitole jsem představil, jak byla hra implementována a jaké principy programování jsem do řešení začlenil. Ve třetí kapitole popisuji, jakým výzvám jsem čelil při vytváření umělé inteligence a jaké druhy jsem nakonec implementoval. Dále si představíme výsledky, ke kterým jsem došel a jak zapadají do vytvoření strategie vhodné i pro lidi.

Bakalářská práce byla psána bez autorských práv na deskovou hru a je určena pouze ke studijním účelům.

1 Pravidla

Z kompletních pravidel hry, které se dají prohlédnout na internetu [[1], jsem použil ty nejdůležitější části, na které se v průběhu práce odkazuji. Dále jsem z pravidel vyčlenil pojmy, které se budou často používat.

1.1 Použité pojmy

Jelikož některé pojmy jsou v kontextu této hry použity jinak, než jaký je standart, je třeba si je definovat.

- **Farmář** – Jde o základní figurku, která slouží k reprezentaci, kolik tahů hráč má a jaký tah udělal.
- **Rodina** – Jedná se o všechny farmáře hráče.
- **Tah** – Jedním tahem hráče se myslí posun farmáře z farmy na akční kartu.
- **Kolo** – Jedno kolo se skládá ze čtyř fází. Nejdříve se přidá nová akční karta, a poté se na příslušných akčních kartách doplní suroviny. Následující třetí fáze je nejdůležitější a zde hráči postupně jeden po druhým hrají své tahy. V poslední fázi se farmáři vrací zpátky na farmu.
- **Sklizeň** – Ke sklizni dochází na konci kol 4, 7, 9, 11, 13 a 14. V období sklizně hráči sklízí suroviny ze svého pole a krmí svoji rodinu. Jestliže hráč nemá dostatek jídla pro každého člena své rodiny, tak obdrží negativní body ve formě žebráka.
- **Karta žebráka** – Karta žebráka je dána hráči za každé chybějící jídlo v období sklizně. Tato karta nemá na průběh hry žádný vliv, ale ve vyhodnocení je počítána za tři negativní body.
- **Farma** – Farma je základní deska, kterou má každý hráč svoji vlastní na začátku hry před sebou a na jakou pokládá získané suroviny a karty.
- **Akční karta** – Sdílené karty mezi všemi hráči, kam mohou hráči umisťovat svoje farmáře. Každá karta specifikuje, jaké akce může hráč udělat po umístění farmáře.
- **Karta profese** – Karty profesí začínají v rukou hráče. Tyto karty se vykládají za použití akční karty a surovin specifikovaných na dané kartě. Efekt vyložených karet je nadále pasivní a hráč na ně nepoužívá další svoje tahy. Možné efekty jsou zlepšení různých akčních karet pro daného hráče nebo získávání dalších surovin v následujících kolech.

- **Karta vybavení** – Vybavení se ve hře rozděluje do malého a velkého. Velké vybavení je ve společné zásobě všech hráčů, zatímco malé je rozdáno hráčům do ruky stejně jako karty profesí. Karty vybavení pomáhají hráči v získávání dodatečných vítězných bodů nebo s obecnou správou farmy. Na rozdíl od profesí nemají efekt na akční karty.
- **Zorané pole** – Kartička zoraného pole značí místo, kde hráč může vysadit obilí nebo zeleninu.
- **Pastviny** – Jedná se o část farmy, která je ohraničena ploty.

1.2 Pravidla hry

Pro jednoduchost jsem se rozhodl popsat pravidla zjednodušeně, abych popsal, co musí hra umět a umělá inteligence musí být schopná vyřešit. Plná pravidla hry se dají najít na internetu [1].

Hra Agricola je určena pro 1 až 5 hráčů. Každý hráč začíná s prázdnou farmou a dvěma farmáři. Během následujících 14 kol hry se snaží svou farmu co nejlépe rozvinout. V každém kole se hráči střídají v postupném posouvání svých farmářů na různé akční karty. Každá karta smí být použita jen jednou v každém kole, takže hráč musí taktizovat i za ostatní hráče, aby věděl, zda mu nějaká akce, kterou potřebuje, nebude zabrána ostatními hráči. V jednom kole hráči hrají postupně jeden za druhým svoje tahy, dokud všichni hráči nepoužijí všechny svoje farmáře.

Na začátku hry se všem hráčů rozdají dva farmáři a dvě chýše, aby je měli kde ubytovat. Jeden hráč dostane žeton, který označuje začínajícího hráče a dostane 2 jídla. Ostatní hráči obdrží 3 jídla. Dále každý hráč dostane 7 karet profesí a 7 karet malého vybavení. Do středu hry se vyloží základní akční karty a karty velkého vybavení.

Kola se skládají ze čtyř fází:

1. Vyložení nové akční karty.
2. Doplnění surovin na akční karty k tomu určené.
3. Práce – Hráči umísťují svoje farmáře na akce.
4. Hráči vrátí své farmáře zpět na svoji farmu.

Na konci kol 4, 7, 9, 11, 13 a 14 dochází k období sklizně, kdy hráči sklidí suroviny ze svých zoraných polí, ze svých zásob vytvoří dostatečné množství jídla za použití velkého vybavení nebo jen přeměny 1 obilí nebo zeleniny za 1 kus jídla. Za

pomocí tohoto jídla nakrmí svou rodinu. Po nakrmení rodiny se jim rozmnoží zvířata, pokud jsou v páru.

Na konci 14. kola končí hra a spočítají se body. Body se dostávají za každé použité políčko na farmě, za množství surovin a farmářů, za různé malé vybavení a za vylepšení jejich ubytování (přechod ze základní dřevěné chýše na chýši jílovou nebo kamennou). Za nedostatek v dané kategorii mohou být hráči body i odečteny. Dále jsou odečteny body hráči, pokud nedokázal nakrmit svou rodinu. Vyhrává hráč s nejvíce body.

Body jsou přiděleny podle tabulky (viz. Tabulka 1) níže. Tabulka popisuje, kolik bodů hráč získá v dané kategorii při dosažení daného množství položek. Tedy za tři krávy získá dva body nebo za šest obilí získá tři body. Dále hráč ztratí jeden bod za každé nepoužité políčko farmy. V neposlední řadě hráč získá bod za každou jílovou chýši, dva body za každý kamenný dům a k tomu získává tři body za každého farmáře. Nakonec se za některé karty vybavení a profesí připočtou další body.

Pokud by tedy hráč měl na konci hry čtyři farmáře v jílové chýši o třech místnostech a splnil každou kategorii alespoň napůl, tak získá 29 bodů a ztratí značné množství bodů za nevyužitá políčka farmy.

Počet bodů	-1	1	2	3	4
Zoraná pole	0-1	2	3	4	5+
Pastviny	0	1	2	3	4+
Obilí	0	1-3	4-5	6	8+
Zelenina	0	1	2	3	4+
Ovce	0	1-3	4-5	6-7	8+
Prasata	0	1-2	3-4	5-6	7+
Krávy	0	1	2-3	4-5	6+

Tabulka 1: Výsledné ohodnocení hráče.

2 Implementace hry

Hra je implementována v jazyce C# ve verzi 7.3 na platformě Windows. Implementace grafického prostředí je za pomocí frameworku Windows Forms. Při návrhu řešení a jeho uskutečnění jsem program rozdělil na uživatelskou, datovou a řídící vrstvu. Uživatelská a řídící vrstva jsou na sebe vázáné přes interface, a tedy nejsou na sobě závislé.

2.1 Struktura programu

Protože nechci dát umělým inteligencím přístup k hlavnímu programu a dát jim schopnost ovlivnit stav hry, tak program je strukturován do komponent tak, aby oddělil nejen logické celky, ale i části logiky určené pro hru a pro hráče.

Program byl tedy rozdělen na samotnou **logiku hry**, která si uschovává aktuální stav hry a jako jediná je schopná ho měnit. Dále na **logiku hráče**, která dostane aktuální stav hry v podobě serializovaných dat a je na ní data interpretovat a rozhodnout se, jaký tah zahraje. V neposlední řadě je zde část programu s nápomocnými funkcemi pro zjednodušení programování.

Rozhraní umělých inteligencí je řešené přes interface, a tedy nová umělá inteligence může být jednoduše přidána jen tím, že daný interface implementuje. Dále je třeba rozšířit konstruktor umělých inteligencí, aby program věděl, jakou umělou inteligenci má vytvořit pro daného hráče při startu hry. Díky tomuto řešení je i grafické rozhraní pro lidi řešené obdobně, a tedy je jednoduše možné zaměnit existující grafické rozhraní za nové nebo dané rozhraní přesunout na internet.

2.2 Použití funkcionálních principů

Jelikož se hra skládá z jednotlivých akcí jdoucích po sobě bez možnosti, aby jeden hráč předběhl druhého a je tedy nemožné, aby dvě akce probíhaly ve stejnou chvíli, tak bylo příhodné v implementaci využít funkcionálních principů. [2]

Nezměnitelnost dat byla jedním z principů, které jsem chtěl využít. Všechny funkce dostanou vždy konzistentní data, která již nikdy nezmění. I jednotlivá volání umělé inteligence je s daty, která se již nebudou měnit, takže jestli si je umělá inteligence někam uloží, tak zůstanou tak, jak jsou. Jen pár konkrétních objektů se

mění a každý z nich má jen jednu komponentu schopnou jej měnit. Konkrétně se jedná o **objekt hry**, který si drží aktuální stav a může se měnit jen zavoláním privátních funkcí, a také o jednotlivé umělé inteligence, které mohou mít svůj vlastní vnitřní stav, kde si drží své mezivýpočty.

Dále jsem začlenil kompozici funkcí, a to do takové míry, že jsem si napsal vlastní rozšiřující funkce, abych byl schopný řetězit volání funkcí bez nutnosti vytvářet komplikované vnořené bloky anebo vedlejší pomocné proměnné. Příkladem může být například funkce **ForEachWithIndex**, která zavolá nad každou položkou v kolekci danou funkci, která bere nejen tu položku jako parametr ale i index dané položky v té kolekci. To umožnilo řetězit a výrazně zpřehlednit mnoho jiných funkcí.

Podobných menších funkcí jsem přidal mnoho, a i když samotné jsou velice jednoduché a často nahrazují velmi základní části, tak dohromady jsou schopné jednoduchým řetězením za sebe a svým jménem popsat kus kódu výrazně přehledněji. Napomohl jsem tomu také vytvořením několika silně typovaných tříd, které přesunuly nutnost ověřovat podmínky na konstruktor. Jedním z daných typů je například **PositiveInt**, který podmínku na pozitivnost má v konstruktoru a tedy, když je předán jako parametr funkce, tak již není třeba ho znovu ověřovat a také se tím docílí jasně daného kontraktu k dané funkci.

Obecně jediné, čemu jsem se z funkcionálních principů snažil vyhnout, byla rekurze, která zaprvé nebyla moc použitelná a zadruhé dělá kód spíše více nepřehledný. Jediné místo, kde ji tedy najdeme, je v implementaci umělé inteligence.

2.3 Reprezentace dat a jejich ukládání

Data jsem se pro jejich čitelnost nejen strojem, ale i člověkem rozhodl ukládat ve formátu JSON. V uložených datech je z každého kola uložen stav hry z konce daného kola. Případně je tam uložen poslední stav kola v případě, kdy je hra uložena v prostředku kola. Dále jsem se rozhodl, pro analýzu odehraných her, ukládat i jednotlivé karty, jak byly hrány za sebou, a jaká část akce byla hrána. Je zde tedy možné ukázat kompletní běh odehrané hry.

Jediný problém, který vzniká ukládáním všech mezistavů, je výrazně zvýšené množství uložených dat a praktická nemožnost se v datech vyznat bez pomocného editoru podporující JSON. V praxi to znamená, že jedna hra má okolo 180 KB neformátovaných dat a po naformátování do čitelné podoby se jedná o 16500 řádků a soubor o velikosti 650 KB.

Zvolený formát je tedy vhodný jen pro zpětnou analýzu a možnost sledovat, jaké rozhodnutí udělala umělá inteligence v daný moment.

Celkově jsem nechal zahrát přes 60000 her a nasbíral jsem přes 16 GB dat. Jelikož se jedná o velké množství dat, tak v příloze práce je složka Data obsahující část uložených her. Konkrétně se jedná o data her algoritmu Monte Carlo, při kterých hrál sám. Všechny uložené hry jsem nahrál na svůj osobní GitHub do veřejné složky, kde si je může každý prohlédnout. [3]

3 Umělá inteligence

Než přejdeme ke konkrétním implementacím, které jsem vytvořil, tak musíme věnovat čas tomu, proč je vůbec potřebujeme. Bohužel i zde narážíme na to, že dokonalá hra se nedá od začátku hry vypočítat a zanalyzovat.

3.1 Problematika dokonalé hry

Kdyby to bylo možné, tak dokonalá umělá inteligence by dokázala následovat dokonalou hru. Tedy by znala všechny možnosti a věděla, jak dojít k dokonalému výsledku. Hra Agricola takové analýze brání více způsoby.

První je množství hráčů. Každý hráč navíc ovlivňuje hru tím, že zaprvé může zabrat akci, kterou je potřeba zahrát k dokonalé hře a zadruhé tím, že s více hráči přibývají karty, které hru do velké míry ovlivňují. Dále akce, které v průběhu kol shromažďují suroviny, jsou sebrány ostatními hráči a nedochází k tak velké akumulaci surovin.

Další důvod, který brání dokonalé hře i ve hře jednoho hráče je náhodnost toho, jaké karty profesí a malého vybavení hráč dostane, a také to, v jakém pořadí přijdou karty na začátku každého kola. Jednotlivé karty jsou omezeny na to, kdy mohou přijít podle období sklizně. Tedy jsou 4 karty, které mohou přijít před první sklizní, 3 karty před druhou, 2 před třetí, a tak následovně. Pro hru jednoho hráče je tedy konkrétně $4! * 3! * 2! * 2! * 2! * 1! = 1\,152$ způsobů, v jakém pořadí mohou být vyloženy prvotní karty akcí, dále hráč dostane náhodných 7 karet profesí ze 14 možných, což dělá 3432 různých kombinací a v neposlední řadě dostane 7 karet malého vybavení z celkového množství 36 karet, což je přes 8,3 milionu kombinací. Ve výsledku se tedy jedná o více jak 33 biliónu různých počátečních pozic, které ve velké míře mohou ovlivnit celkový průběh hry. Množství různých počátečních pozic roste s množstvím hráčů tím, jak přibývá i množství karet. V rámci krátkého pozorování jsem si všiml, že hráč má v průměru 10 různých akcí, které může zahrát v jednom tahu a ty většinou mají mnoho podmožností, které mohou být hrány. Ve výsledku se tedy jedná o množství pozic a možností, které není možné v rozumném čase projít.

3.2 Použité implementace

V práci jsem se rozhodl použít 3 typy umělých inteligencí, které na sebe logicky navazovaly a navzájem se doplňovaly. **Náhodný hráč** fungoval jako základ všech

umělých inteligencí. **Monte Carlo tree search** nebyl limitován neexistencí hodnotící funkce, která by byla schopná ohodnotit hru v jejím průběhu. Na poznatcích z Monte Carlo algoritmu jsem vytvořil strategii hry a přesunul jsem ji do hodnotící funkce pro **MinMax algoritmus**.

3.2.1 Náhodný hráč

Náhodný hráč vezme seznam všech validních a hratelných karet a vybere z nich jednu náhodně. Pokud vybraná karta má více možností, jak může být hrána, tak z nich vybere náhodně jednu. Mezi takové případy spadají jak karty obsahující více akcí, které ne vždy musí být zahrány všechny, tak i karty, které dávají hráči více možností, jak danou akci využít. Tedy každá karta má stejnou pravděpodobnost, že bude vybrána bez ohledu na počet způsobů, jakým může být hrána. Důvod pro toto oddělení byl ten, aby akce, které jen získávají suroviny, a tedy mají jen jeden způsob zahrání, měly stejnou šanci být hrány, jako karty, které způsobů mají více. Konkrétní příklad může být karta s akcí **Profese**, která na začátku hry může být hrána 7 způsoby (počet karet profese v ruce hráče).

Tato umělá inteligence nedosahuje vysokých výsledků, protože i závěrečná kola, která jsou nejdůležitější, jsou hrána náhodně bez snahy o vyšší skóre. Ovšem byla užitečná k ověření fungování programu a jako základ pro následující umělé inteligence, které buď využívaly implementaci samotnou pro simulaci hry protihráčem, nebo využívaly přidružené funkce jako je seznam všech dostupných tahů.

3.2.2 Monte Carlo tree search

Monte Carlo [4] byla použita především proto, že pro tuto hru neexistuje ocenění obecné pozice v průběhu hry, ale pouze až na konci hry, což je situace, která tomuto algoritmu nevadí, a naopak je to vše, co tento algoritmus potřebuje.

Tato umělá inteligence tudíž dosahovala výrazně vyšších výsledků než předchozí algoritmus, jelikož se snažila vždy jít cestou za vyšším skórem, což v závěrečné části hry vedlo k vyšší optimalizaci skóre.

Díky této umělé inteligenci jsem byl schopný poskládat smysluplné výsledky a dojít ke hrám s dostatečně vysokým skóre, že už by je bylo těžké porazit. Dále se

začaly objevovat trendy, jaké karty jsou lepší, v jakém momentu hry, což vedlo k možnosti vytvoření strategie pro lidského hráče.

Algoritmus Monte Carlo funguje tak, že vezme seznam všech možných akcí a pro každou z nich vytvoří simulaci hry, která vychází ze stavu po zahrání dané akce. V dané simulaci nechá za sebe a za ostatní protihráče hrát náhodný algoritmus (tedy náhodnými kroky dojde ke konci hry) a vyhodnotí, jaké skóre se v takovém případě podařilo získat. Aby výsledná data byla spolehlivější, tak simulaci pro každou akci provádí vícekrát. Poté vybere akci s nejvyšším průměrným skóre a provede simulace pro následující tahy. Výsledky nových simulací propaguje zpět k tahům, které k nim vedly. Tímto způsobem vytváří strom simulovaných akcí a jejich průměrné skóre. Na konci vybere tu větev stromu, která s největší pravděpodobností povede k nejlepšímu výsledku. Tedy vybere tu akci, které měla nejvyšší průměrné skóre.

3.2.3 MinMax algoritmus

Použití MinMax [5] algoritmu sloužilo k ověření navrhovaných strategií pro hru lidským hráčem. Strategie tedy musela být popsána nikoli typem akce, který chce hráč zahrát, ale jaké karty je lepší hrát v jaké fázi hry. Například se tedy neřekne, že hráč chce vybudovat trvalý zdroj potravy, ale musí se říct, že hráč chce hrát karty, ze kterých získává dobytek a karty, které mu umožní zasít obilí nebo zeleninu.

Data pro tento algoritmus jsem navrhnul tak, aby bylo možné dávat všem kartám ve hře různé priority v závislosti na tom, jaké kolo se zrovna hraje. Algoritmus jsem zároveň neomezil jen na karty, ale i na počet farmářů, množství různých surovin a do jaké míry se má zabývat finálním skóre. Jinými slovy je možné říct, že některé karty jsou lepší na začátku hry, kdy je důležité získat farmáře navíc, na konci hry zase karty, které dají nejvyšší skóre.

Mnou napsaný MinMax tedy funguje tak, že si drží tabulku ohodnocení karet a všech výše popsaných kritérií. Vezme všechny dostupné tahy a pro každý tah vezme všechny karty dalšího hráče. Takto počítá do konce kola (respektive do svého posledního tahu v daném kole) všechny možnosti a všechny je ohodnotí pomocí daných kritérií a vytvoří tím ohodnocený strom. Zároveň si u každé akce drží, jestli se jedná o tah jeho nebo tah protihráče. Tím vytvoří strom, jehož listy jsou ohodnocené a každý uzel si pamatuje, o čí tah se jedná. Poté od listů každý předcházející uzel vezme minimální nebo maximální hodnotu ze všech jeho

následovníků. Minimální hodnotu berou uzle, které jsou protihráčovo a maximální hodnotu berou uzle, které jsou jeho.

Hodnotící tabulku pro MinMax jsem se rozhodl ukládat ve formátu JSON. Zaprvé to umožnilo jednoduchou čitelnost jak strojem, tak i člověkem. A za druhé to je formát, který jsem už používal pro ukládání hry, a tedy jsem chtěl být dále konzistentní.

Současnou hodnotící funkci jsem nechal vytvořit z dat, která jsem získal z Monte Carlo her. Vzal jsem jednotlivé karty, které byly hrány v každé hře a pamatoval jsem si, **v jakém kole byly zahrány** a **jakého skóre dosáhly**. Tak jsem byl schopný vypočítat, jakého průměrného skóre každá karta dosáhla v jakém kole. Aby výsledná hodnotící funkce nebyla závislá na rozdílu ve skóre mezi koly, tak jsem pro každé kolo zvlášť a pro každou kartu v tom kole vypočítal, jaký procentuální zisk má z celkového skóre v tom kole. Tedy, pokud bych měl čtyři karty s průměrným skóre v jednom kole 3, 3, 2, 2, tak první a druhá karta budou mít 30 % a třetí karta se čtvrtou budou mít každá 20 %. Abych nemusel řešit velmi malá čísla, tak jsem každé číslo nechal v procentuální podobě (tj. 30 místo 0,3) a všechna čísla jsem vynásobil 10. Pro program toto umělé zvětšení čísel konstantou nemá vliv, jelikož při počítání nejlepšího tahu budou všechny výsledky vynásobeny stejnou konstantou. To vychází z toho, že nejlepší tah se počítá z porovnávání dvou čísel mezi sebou a operace porovnání je nezávislá na vynásobení kladnou konstantou.

3.2.3.1 Alfa-Beta MinMax algoritmus

I při omezení výpočtu jen na konkrétní kolo docházelo k extrémně dlouhým dobám výpočtu. K tomu docházelo proto, že i když byli ve hře jen dva hráči, tak se dohromady jednalo o minimálně čtyři farmáře. Každý farmář měl průměrně deset možných akcí, a tedy jen pro ně algoritmus musel vypočítat kolem $10^{\#farmářů\ v\ kole} = 10^{2+2} = 10000$ možných pozic pro každý tah a s každým farmářem navíc se to výrazně zhoršovalo. Uvažoval jsem o dvou způsobech, jak tento problém vyřešit.

První bylo analyzovat, jaké tahy se hrály a pokud daná pozice už existovala z předchozích výpočtů, tak dále nepočítat. Tady jsem narazil na několik problémů, které dané řešení dělalo nereálné. Jedním z problémů bylo, jak určit, že daná pozice už existuje. To, že farmáři jsou na stejných akcích, neznamená, že tam došli ve stejném pořadí a mnoho akcí se bude chovat a vyhodnocovat jinak v závislosti na

pořadí. Například když hráč zahraje akci pro zisk dřeva a akci pro postavění plotů v tomto pořadí, tak má více dřeva na stavbu plotů, a tedy akce je sama o sobě lepší, než kdyby pořadí zahraných karet prohodil. Druhým velkým problémem bylo, že daná analýza i tak pomáhala až u druhého tahu farmáře u daného hráče. Tedy hráči museli mít více jak dva farmáře, aby se vůbec nějaký čas zkrátil.

Druhým řešením bylo do implementace přidat takzvaný Alfa-Beta ořezávání. [6] Jedná se o omezování prostoru možností pomocí porovnání s nejlepším současným výsledkem. Tedy během výpočtu si držíme v každém uzlu v proměnné alfa nejlepší doposud dosaženou hodnotu v uzlu našeho hráče a v proměnné Beta doposud nejnižší nalezenou hodnotu v uzlu protihráče. Poté když zjistíme, že uzel našeho hráče má vyšší hodnotu, než je hodnota Beta, tak přestaneme zbylé části možnosti dané větve procházet a počítat. To můžeme udělat proto, že víme, že daný hráč by nezahrál tah, ve kterém by náš výsledek byl lepší než v jiné možnosti, kterou může zahrát. Obdobně to platí pro uzle protihráče, jen porovnáváme s hodnotou Alfa a zajímá nás, zda existuje uzel protihráče s hodnotou menší, než je Alfa.

Celé to vychází z toho, že daná minima a maxima jsou propagována skrze jednotlivé uzle odspodu. Tedy pokud se jedná o uzel protihráče, tak jeho hodnota nikdy nestoupne nad hodnotu libovolného následujícího uzle. Víme tedy, že pokud je tato hodnota menší než nejhorší možný výsledek (hodnota Alfa), tak nemá smysl zbytek podstromu počítat, jelikož výsledek už nebude pro nás lepší než tento špatný výsledek. Obdobně to funguje pro naše uzle a hodnotu Beta.

Jedinou potřebnou změnou v programu oproti klasickému MinMaxu bylo počítat hodnoty uzlů už od samého začátku, držet si hodnoty Alfa a Beta a ve vhodných místech zarazit výpočet zbytku stromu.

Toto zrychlení je tedy nejlepší v případech, kdy se finální nejlepší řešení najde velmi brzy. V případech, kdy se najde až v poslední větvi, tak nedojde k žádnému zrychlení, ale spíše k zanedbatelnému zpomalení kvůli propočtu Alfa Beta proměnných a vyhodnocování podmínek. V opačném případě, kdy se optimální výsledek najde hned v první větvi, tak zbytek stromu bude velice zredukován, jelikož zarazíme mnoho výpočtů hned při průchodu prvním uzlem.

4 Výsledky hráčů

Nejprve bych rád popsal, jakého výsledku se snaží umělé inteligence dosáhnout. Bohužel neexistuje žádná oficiální statistika, a tedy nebylo možné, abych vycházel z velkého množství her. Na základě mnou odehraných her, kterých bylo okolo dvaceti, jsem vypozoroval, že běžné skóre potřebné na vítězství se pohybuje okolo 25 bodů. Dané rozmezí bodů potřebné k výhře je mezi 22 a 32 body, v závislosti na počtu bodů protihráčů. V extrémních případech se nám podařilo překonat i 40 bodů. Normální výsledky v námi hranými hrami byly převážně vždy nad 15.

4.1 Porovnání mezi různými typy hráčů

Nepřekvapivě **náhodný hráč** má v porovnání s ostatními hráči nejhorší výsledky. V průměru dosahoval skóre 8,48. Jedná se o algoritmus, který bez uvážení vybírá náhodné možnosti a neřeší, jaký finální počet bodů bude mít.

MinMax algoritmus, který měl vyhodnocovací funkci založenou čistě na výsledcích průměru z Monte Carla, nedosahuje tak velkých výsledků jako Monte Carlo algoritmus. Přestože jsem mu přidal do vyhodnocovací funkce, aby se ke konci hry soustředil převážně na body, tak to nestačilo. Průměrné skóre bylo v jeho případě 10,96. Důvodem je, že i Monte Carlo se z principu jeho algoritmu soustředí ke konci hry na body defacto stejně. Výsledné skóre hry z MinMax algoritmu ve hře jednoho hráče bylo výrazně konzistentnější oproti ostatním algoritmům tím, že zde už náhoda figuruje jen v tom, jak hra začne.

Monte Carlo algoritmus dosahoval nejvyšších výsledků celkových i průměrných. Na rozdíl od MinMax algoritmu není vázaný na předem dané vyhodnocovací funkce a je výrazně pružnější. Tedy, když přijdou karty akcí v jiném pořadí nebo když je zrovna potřeba získat trochu více jídla v konkrétní hře, tak je schopný se přizpůsobit a konkrétního cíle docílit.

Přestože algoritmus náhodného hráče měl nejvíce odehraných her, tak dosáhl maximálního skóre 33 a her se skóre vyšším jak 25 bylo jen 1,31 %. Monte Carlo oproti tomu došlo až ke skóre 38 a her se skóre vyšším jak 25 mělo přes 31 %. Samotný MinMax algoritmus pak dosahoval nižšího skóre a nedostal se přes skóre 25.

Kvůli časové náročnosti algoritmu MinMax bylo provedeno čtyřikrát méně her než her pro algoritmus Monte Carlo. Kvůli množství různých počátečních pozic a počtu odehraných her, je třeba brát výsledná data s uvážením statistické odchylky, a to převážně u Maximálního skóre.

Jméno algoritmu	Maximální skóre	Průměrné skóre	Podíl her nad 25
Náhodný hráč	33	8,48	1,31 %
Monte Carlo	38	22,32	31,02 %
MinMax	23	10,96	0 %

Tabulka 2: Porovnání výsledku algoritmů.

4.2 Vyvození základních poznatků z výsledků

Při analýze výsledků z jednotlivých her jsem se snažil vyvodit, z jakého důvodu k vyššímu skóre došlo. K analýze jsem použil výši průměrného skóre, kterého daná karta dosáhla při zahrání v konkrétním kole.

Všiml jsem si, že úspěšné hry se už od začátku snažily postavit další chýši a mít jen tak tak jídla na první sklizeň. Dále většina úspěšných her se snažila ve středu hry stavět ploty a sázet obilí nebo zeleninu. Konce hry bývaly celkem náhodné a vypadalo to, že hráči prostě získávali to, co jim chybělo. V nejvyšším množství se jednalo o získání kamene, který je jinak během hry velmi vzácnou a nedostatkovou komoditou. I když z analýzy her vyplývalo, že kámen je nejlepší na konci hry pro získání nejvyššího skóre, tak jsem věděl, že kamenem samotným to není, jelikož za něj žádné body nejsou. Kámen se ovšem používá jak na některá velká nebo malá vybavení, která přidají body, tak i na renovaci chýše na kamennou, což přidá značné množství bodů.

4.3 Shrnutí strategie

Z pozorování jsem vyvodil, že MinMax není nejlepší algoritmus v takto komplikované hře, jelikož není schopen se dostatečně přizpůsobit současným podmínkám. To je minimálně pravda v jeho základním stavu, který vznikl jen na základě průměrů a který nevyužívá další možnosti vyhodnocovací funkce.

Z her a z předešlých poznatků jsem vyvodil, že nejlepší hry byly ty, kdy se hráč pokusil mít k pátému kolu nového farmáře a pro ně nějaké jídlo získané

z jakéhokoliv zdroje. Tedy musel získat dostatek surovin na postavení nové chýše, postavit chýši a vyprodukovat farmáře.

Ve střední části hry se hráč snaží získat trvalý zdroj potravy a obsadit zbytek svojí farmy chlévy, farmami a pastvinami. Trvalý zisk potravy je lepší dělat různorodě, tedy mít různá zvířata alespoň v párech, mít zaseté obilí a zeleninu. Za každou z těchto surovin je ve finální části významné množství bodů a zároveň to zajistí hráče k tomu, aby v pozdější části hry už nemusel řešit dostatek obživy a mohl se soustředit jen na zisk bodů.

Ve finální části hry se hráč soustředí převážně na vyplnění chybějících částí a tím co největší zisk bodů. Mezi takové akce patří získávání chybějících zvířat, renovace chýše a zisk dalšího farmáře většinou za použití akce potomstva i bez volné chýše.

4.4 Implementace strategie

S přihlédnutím k předešlé kapitole jsem se pokusil přepsat danou strategii do mé implementace hodnotící tabulky, a tedy i do funkce pro MinMax algoritmus.

První změnou byla změna hodnocení karet tak, že jsem vynechal hodnocení těch nedůležitých. Naopak jsem zachoval nebo zvýšil hodnocení u karet, které se týkaly popisu strategie.

Druhou změnou bylo přidání hodnoty k ohodnocení počtu farmářů a to tak, že v pátém kole se snaží co nejvíce získat jednoho farmáře navíc. K motivaci jsem přidal dostatečné skóre i k docílení farmáře navíc dříve, ale motivoval jsem ho méně body, aby se nepokusil o jeho získání před první sklizní. Před první sklizní je těžké získat farmáře navíc a všechny nakrmit, a tedy je velké riziko přidělení záporných bodů. Poté jsem dal za získání čtvrtého farmáře menší ohodnocení, aby bylo výhodnější mít jen tři. Tak to platí až do konečné fáze hry, kdy od 12 kola je za to naopak vysoké skóre.

Abych hráče donutil myslet na jídlo, tak mou **třetí změnou bylo přidání vysokého skóre za vlastnění jídla**. Převážně v začátku hry, tedy ještě před první sklizní, by se měl snažit o to, aby měl k dispozici dostatek jídla. Později ho už motivuji méně, jelikož tou dobou by už měl mít rozjeté farmy a přípravu na velké generování jídla, a tedy není potřeba stínit ostatním položkám.

Čtvrtou změnou byla motivace se soustředit na finále hry. Ke zvyšování skóre do finále hry a do počítání výsledných bodů ze hry ho motivuji od desátého kola. Z počátku pomalu, ale ve dvanáctém a v posledním kole motivaci extrémně zvyšuji.

To až do takové míry, že v posledním kole to je to jediné důležité a převyšuje to veškeré ostatní činnosti.

4.4.1 Výsledky strategie

Hodnotící tabulku, kterou jsem slovně popsal v předešlé části, jsem vyzkoušel ve hře samotné a ve hře proti ostatním hráčům. MinMax algoritmus vybavený touto tabulkou překonal výsledky předchozí tabulky a zároveň dosahuje téměř na hodnoty Monte Carla. Konkrétně tedy dosahoval finálního skóre přes 25 a to přibližně v 20,8 % případů. Průměrné skóre hry bylo 18, tedy výrazně vyšší než u předchozí tabulky a jen o málo nižší než u Monte Carla.

Jméno algoritmu	Maximální skóre	Průměrné skóre	Podíl her nad 25
Náhodný hráč	33	8,48	1,31 %
Monte Carlo	38	22,32	31,02 %
MinMax	23	10,96	0 %
Strategie hráče	31	18	20,83 %

Tabulka 3: Porovnání výsledku algoritmů a navržené strategie.

Bohužel MinMax algoritmus stále bojuje s tím, že sám není velmi přizpůsobivý současné situaci. V případě potřeby se nerozhodne zahrát akci jinak jen proto, že má na ruce zajímavou kartu profese nebo malého vybavení, protože tak to nemá ohodnocené. Zatímco Monte Carlo se nebojí vyzkoušet všechno a případně odejít od toho, co by udělal normálně. Na druhou stranu je tato strategie více konzistentní, takže i když nenajde lepší možný výsledek, tak ani nespadne do výrazně horšího výsledku.

Nakonec jsem nechal hrát proti sobě hru s algoritmy Monte Carlo, MinMax a naší strategii. Zde se výrazně projevila nepřizpůsobivost MinMax algoritmu k současnému stavu hry tím, že průměrné skóre šlo dolů výrazněji než u algoritmu Monte Carla. Ještě více se tato vada projevila u naší strategie, která se soustředí na správné věci, ale neví, co má dělat v momentě, kdy jsou všechny potřebné karty obsazeny. To se projevilo tím, že vyhrává méně často než MinMax algoritmus, i když naše strategie dosahuje lepších průměrných výsledků (viz. Tabulka 4).

Pro úplnost dat chci doplnit informaci, že za vítězství pro daného hráče jsem považoval stav, kdy daný hráč má největší skóre. Tedy případ, kdy dva hráči měli stejný nejvyšší výsledek, se počítá jako výhra pro oba.

Jméno algoritmu	Maximální skóre	Průměrné skóre	Poměr vítězství
Monte Carlo	24	12,18	63,64 %
MinMax	17	4,27	22,73 %
Strategie hráče	20	6,54	18,18 %

Tabulka 4: Porovnání hry Monte Carla, MinMax a naší strategie.

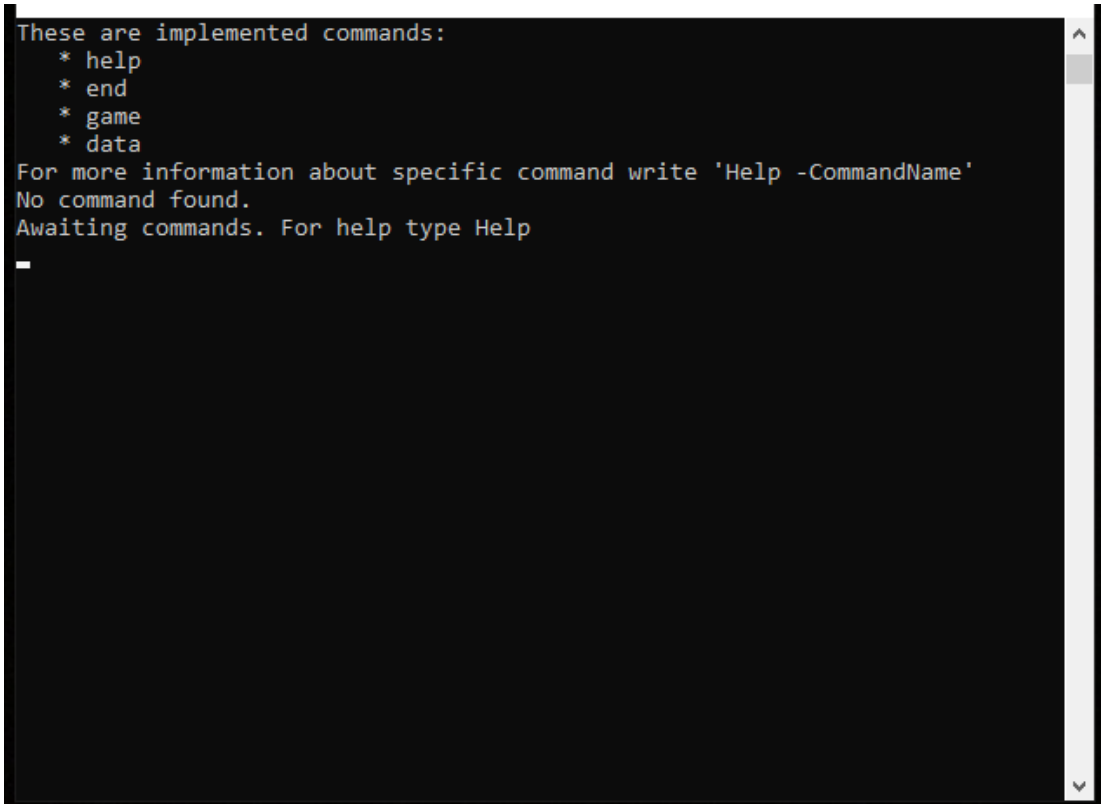
5 Uživatelská dokumentace

Pro správné spuštění programu se vyžaduje systém Windows 10 od firmy Microsoft z důvodu využití běhové podpory CLR. Program lze spustit ve dvou režimech. První režim spouští program v příkazovém řádku a druhý ho pouští v grafickém prostředí. Oba zkompilované režimy lze najít v připojeném .zip balíku Agricola ve složce Raw. První je ve složce pojmenované CMD a druhý je ve složce GUI.

Jelikož cíl práce byl především vývoj výpočtových algoritmů a sestrojení strategie na základě poznatků, tak grafické prostředí je velmi omezené jak v množství funkčnosti, tak i v její aplikaci. Například není možné vybrat, proti jakému algoritmu bude hráč hrát.

5.1 Režim příkazového řádku

Po spuštění režimu v příkazovém řádku se uživateli vypíšou všechny dostupné funkce.



Obrázek 2: Spuštění programu v příkazovém řádku.

Uživatel volá funkce ve formátu „názevFunkce -parameter1 -parametr2”. Každá funkce má dostupné jiné parametry, které modifikují nebo specifikují její chování. Po

napsání funkce ve správném formátu je třeba zmáčknout Enter, aby se daná funkce vyhodnotila. V případě potřeby je možné za sebe napsat více funkcí. V takovém případě budou funkce volány v daném pořadí a další funkce bude zavolána vždy, až když je funkce před ní dokončena. Toto se dá využít pro naplánování procesů, a tedy odstraňuje to nutnost mít uživatele přítomného při konci jedné funkce, aby mohla být zavolána funkce další. Příklad tohoto využití může být spuštění hry pro jeden druh umělé inteligence, poté pro druhý a na konec vyhodnocení výsledků obou programů.

Dostupné parametry a jejich vysvětlení se dají získat zavoláním příkazu Help a s názvem funkce jako její parametr. Příkaz Help je schopný převzít více parametrů najednou a vypíše popis ke všem z nich.

```
help -help -game -data
Command help will print all available commands. If provied with argument in
 type of another command. It will write additional information about it.
Command game will start the game played by computer
    By default the game will be created with random number of players and w
ith randomly selected AI
    By providing additional arguments it is possible to change the setting.
 All non defined arguments will fallback to its default value
    -pa{%d} will start %d number of games in parallel. By default this is 1
. Usage: "game -pa3" will start 3 games in parallel
    -ai{name}{%d} will assgin type of AI(name) to player %d. Players are in
dexed from 1. By default it is selected randomly
    -pc{%d} will set the number of players. By default it is randomly selec
ted
    -ga{%d} Number of games played.
Command data will process all files in the provided folder
    -{folderName} add name of the folder in the SaveFolder to process file
in this folder
No command found.
Awaiting commands. For help type Help
```

Obrázek 3: Seznam všech příkazů a jejich parametrů s vysvětlením.

5.1.1 Příkaz Game

Zavolání příkazu game má za následek spuštění hry ve vedlejším vlákně. Parametry můžeme ovlivnit, jaké umělé inteligence budou proti sobě hrát, kolik hráčů ve hře bude, kolik her se má odehrát a kolik procesorových vláken se může k výpočtu využít.

Po zavolání se hry pustí a do příkazového řádku jsou vypisovány základní informace o zpracovaných hrách. Mezi danými informacemi vidíme například

kolikátá hra byla spuštěna, s jakými algoritmy daná hra běží, jak je uložený soubor s průběhem hry pojmenován a jak dlouho daná hra trvala.

```
Awaiting commands. For help type Help
game -pc2 -aiRandom1 -aiRandom2 -ga90 -pa5
Starting 5 games in parallel
Created game number 1 with players: Random Random
Created game number 2 with players: Random Random
Created game number 3 with players: Random Random
Created game number 4 with players: Random Random
Created game number 5 with players: Random Random
Final score 13
Final score 0
Final score -8
Final score -7
Final score -1
Saving game with name finished-_-2-_-20230509-_--1-_-74a9b8bd-9307-41d9
-85b4-94feaeb7ef93 after 0,1695548 seconds
Saving game with name finished-_-2-_-20230509-_-0-_-ae2329e2-04b7-4693-
918b-d8fe69577e6a after 0,1689529 seconds
Saving game with name finished-_-2-_-20230509-_--8-_-ad80f4e8-3c4d-48c5
-b4a8-69c98299a10f after 0,1692388 seconds
Saving game with name finished-_-2-_-20230509-_-13-_-a25fbc0d-b329-424d
-8e9b-1c81716efaf7 after 0,1683881 seconds
Saving game with name finished-_-2-_-20230509-_--7-_-713d4ecb-33cb-44a1
-b3b2-93c01c6e804a after 0,1693815 seconds
Created game number 6 with players: Random Random
Created game number 7 with players: Random Random
Created game number 8 with players: Random Random
Created game number 9 with players: Random Random
Created game number 10 with players: Random Random
Created game number 10 with players: Random Random
Final score 6
Saving game with name finished-_-2-_-20230509-_-6-_-24fd8969-8aff-48ae-
8972-af64bbe9fc27 after 0,0784852 seconds
Created game number 11 with players: Random Random
Final score -5
```

Obrázek 4: Ukázka výpisu informací při spuštění více her najednou.

5.1.2 Příkaz Data

Tento příkaz slouží k analýze dat z odehraných her. Jeho jediným parametrem je název složky, z které se mají data získávat. Ze všech her v dané složce vezme průběh hry a vytvoří seznam karet, které byly daným typem umělé inteligence hrány, a spočítá, jakého průměrného výsledku daná karta dosahuje.

Tuto analýzu lze použít pro sestrojení nebo úpravu MinMax vyhodnocovací tabulky.

5.2 Režim grafického okna

Po spuštění v režimu grafického okna se uživateli zobrazí hlavní nabídka s možnostmi vytvoření nové hry, načtení uložené hry a možnost spustit hru hranou umělou inteligencí.

Obrázek 5: Hlavní menu v grafickém okně.

5.2.1 Nová hra

Po kliknutí na tlačítko s nápisem Nová hra, je uživatel přesunut na obrazovku s výběrem, proti kolika soupeřům chce hrát. Každý takto přidaný protihráč bude používat algoritmus náhodného hráče.

Po vybrání počtu soupeřů musí hráč kliknout na tlačítko Nová hra v pravém dolním rohu. Po kliknutí je přesunut na obrazovku s vlastní hrou.

Obrázek 6: Okno nové hry s výběrem množství protihráčů.

5.2.1.1 Obrazovka s probíhající hrou

Uživateli se zobrazí obrazovka se hrou samotnou. Zde se nachází osm různých bodů zájmu.

Obrázek 7: Okno z probíhající hry s označenými body zájmů.

Danými body jsou:

1. Místo, kde se nachází karty akcí.
2. Část plochy, kde jsou uloženy karty velkého vybavení.

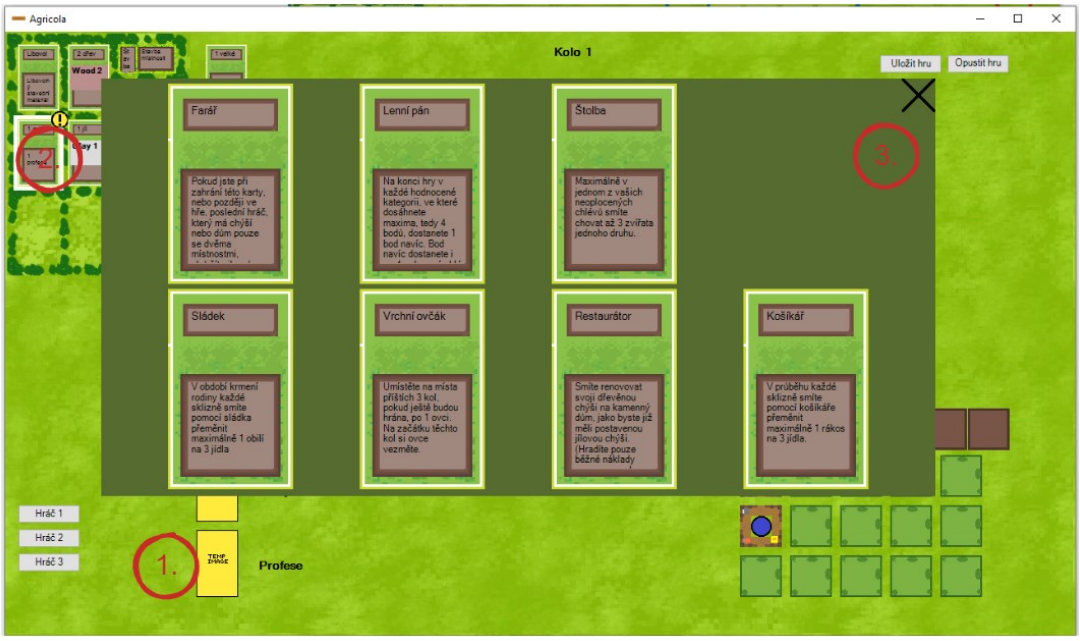
3. Plocha s farmou aktuálně zvoleného hráče.
4. Karty malého vybavení a karty profesí, které má hráč v ruce.
5. Již zahrané karty profesí, malého vybavení a velkého vybavení daného hráče.
6. Tlačítka, která přepínají, jakého hráče bude vyobrazena farma.
7. Číslo současného kola.
8. Možnost hru ukončit nebo ji uložit.

Modrá kolečka v bodě zájmu 3 značí současné umístění farmáře. Chce-li hráč farmáře posunout, tak musí kliknout na akční kartu z bodu zájmu 1 dvakrát. Po prvním kliknutí se objeví bílé ohraničení kolem karty a po druhém bude na kartu farmář přesunut.

Zde bych chtěl upozornit, že daný program nebyl určen k přesné reprezentaci reálné hry a tedy to jak bude daná akce zahraná, je vybráno náhodně ze všech možností.

Obdélníky kreslené přes karty v jednotné barvě s nápisem a číslem reprezentují, jaké suroviny a kolik jich je na dané kartě umístěno.

Aby uživatel mohl vidět, jaké karty má v ruce, a nebo když by potřeboval zvětšit napsané informace na kartě, tak musí kliknout na danou žlutou kartu balíčku. Případně pro zvětšení informace o kartě, musí na danou kartu nejdříve kliknout a poté kliknout na vykřičník v žlutém kolečku, který se objeví. V ten moment se přes obrazovku objeví rámeček obsahující seznam karet v ruce, případně informace o dané kartě. Tento rámeček se pak křížkem v pravém horním rohu zavírá.



Obrázek 8: Vyobrazení profesí v ruce hráče.

5.3 Načíst hru

Po zmáčknutí tlačítka Načíst hru se uživateli zobrazí obrazovka s vypsaným seznamem všech uložených her. Zde musí uživatel na jedno z jmen kliknout a pak může kliknout na tlačítko Načíst hru. To ho přesune do obrazovky s buď probíhající hrou nebo do obrazovky s dokončenou hrou, která je popsaná níže.

Obrázek 9: Menu pro načtení uložené nebo dokončené hry.

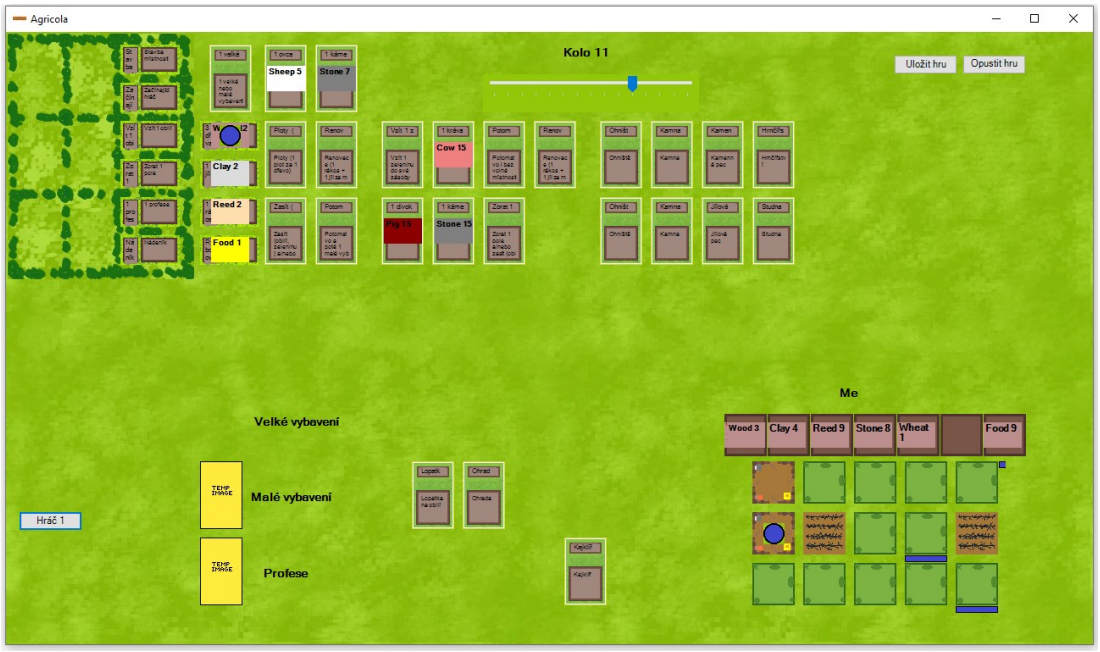
Hry jsou pojmenovány podle počtu hráčů ve hře, současné kolo hry a datum vytvoření hry v tomto pořadí.

5.3.1 Obrazovka s dokončenou hrou

Tato obrazovka je ve velké míře shodná s obrazovkou s nedokončenou hrou. Jediný rozdíl je v tom, že v dané hře už dvojklik karty nic neudělá a pod číslem kola se nachází posuvník, kterým se vybírá, jaké kolo se má zobrazit.

Limitací tohoto zobrazení je, že všechny karty akcí jsou viditelné i přes to, že je hráč v té době zahrát nemohl. Také reprezentace stavu hry v daném kole je z jeho konce, tedy po zahrání posledního farmáře.

Jedná se jen o orientační přehled současného stavu, který nemusí být naprosto přesným.



Obrázek 10: Okno s dokončenou hrou.

5.4 Hra umělých inteligencí

Na této obrazovce může uživatel spustit hru různých umělých inteligencí ve hře proti sobě. Pro každého až z pěti možných hráčů může vybrat, jaký algoritmus má hráč využívat a nebo zda ve hře vůbec má být.

Poté kliknutím na tlačítko Spustit hru se vytvoří hra s danými hráči a spustí se na pozadí. Zde musí uživatel počkat, dokud se mu pod tím tlačítkem neobjeví nápis, který oznamuje, že hra byla uložena. V ten moment se mu pod tímto nápisem objeví

tlačítko **Načíst hru**, které mu umožní danou hru okamžitě načíst. To uživatele přesune do obrazovky s dokončenou hrou popsanou výše.

Obrázek 11: Menu pro výběr a spuštění hry umělých inteligencí.

6 Programátorská dokumentace

Jako platforma pro tento program byl zvolen systém Windows. Veškerý kód je psaný v jazyce **C# verze 7.3** za použití frameworku **.Net Framework 4.7.2**. Veškerý grafický výstup je psaný pomocí frameworku **Windows Forms**. Starší verze C# byla použita kvůli tomu, že Windows Forms novější nepodporují.

Pro vývojový editor jsem použil **Visual Studio 2022**, který má vysokou podporu pro vývoj v C#. Při programování jsem využil dokumentaci C#, .NET a Windows Forms dostupných na webu [7].

Daný program je napsán jako jeden projekt a je spustitelný ve dvou režimech. Prvním je režim grafického výstupu a druhým je režim z příkazového řádku. Součástí projektu je nejen hra samotná, ale i doprovodná logika pro analýzu dat.

Jednotlivé logické celky jsou seskupeny do společné složky. Tedy veškerý kód k analýze dat je ve složce **DataAnalyzer**. Veškeré pomocné funkce k běžnému programování jsou ve složce **Extensions** a obdobně pro ostatní části.

Mimo složky je zde důležitý soubor **Program**, který funguje jako počáteční bod celého řešení a je zodpovědný za vytvoření grafického okna nebo za otevření příkazového řádku. Dále se zde nachází **MenuController** a **ConsoleController**. **MenuController** zodpovídá za správné chování grafického okna. Přepíná jednotlivé obrazovky a v případě potřeby aktualizuje jejich obsah. Oproti tomu **ConsoleController** zpracovává příkazy na příkazovém řádku a na jejich základě vyvolává dané akce. Základním příkazem pro **ConsoleController** je příkaz **Help**, který vypíše seznam všech možných příkazů a k nim jejich popis a specifikace.

Rád bych se ale věnoval jen těm nejdůležitějším.

- **Game** – V této složce se nachází veškeré věci, které se týkají vnitřního fungování hry. Jedná se o definici karet až po zpracování hráčova tahu.
- **GameAI** – Zde nalezneme všechny naše implementace algoritmů pro umělou inteligenci. Zároveň je zde i místo, kde se nová umělá inteligence musí registrovat, aby bylo možné ji použít pro hru.
- **Menus** – Do této složky jsem vložil definice všech grafických oken a jejich logiku.
- **Extensions** – Jelikož je hodně kódu, který by byl repetitivní nebo hodně kroků, které provádíme často nad různými datovými typy, tak jsem do této složky vložil veškeré pomocné funkce, které nám pomohou kód zpřehlednit.

- **Serialization** – Je poslední hlavní složka řešení. V ní jsou veškeré potřebné funkce k ukládání a načítání dat ze souboru. Funkce na přeměnu dat z formátu JSON do objektů a opačně.

6.1. Game

Tato složka obsahuje vše, co je potřebné k samotnému běhu hry. Sama má v sobě další složky, které obsahují myšlenkově podobné soubory. Jedná se o složky:

- **Dto** – Obsahuje datové struktury reprezentující veškeré objekty hry v primitivních datových typech. Tyto struktury jsou pak ty, které jsou ukládány, načítány nebo předávány mimo složku **Game**. Změna na primitivní datové typy byla potřeba kvůli pozdějšímu uložení ve formátu JSON. Zároveň to umožňuje oddělení skutečných dat hry od dat, které se předávají zbytku programu, a tedy nehrozí, že dojde k jejich nechtěným změnám.
- **Resources** – Tato složka obsahuje obdobně jako Dto veškeré datové struktury potřebné pro hru. Je zde tedy struktura **Game**, **Farmer** a další. Všechny tyto struktury jsou převoditelné do svých Dto pomocí implementace interfacový metody **Serialize**.

Má-li soubor ve jménu slovo **Controller**, pak se jedná o jediný soubor, který by měl napřímo manipulovat s přiřazenou třídou.

Hlavním souborem složky **Game** je tedy soubor **GameController**, který obsahuje veškeré funkce k činnosti hry. Jedná se, jak o zakládání nové hry nebo o načítání existující hry, tak i o provádění jednotlivých tahů a o získávání dalších tahů od hráčů. Také zde můžeme najít funkci, která spustí celkovou hru a stará se o celý její průběh. Jedná se tedy o jediný soubor, který přímo manipuluje s datovou strukturou **Game**.

Další důležitý soubor je soubor **CardConstructor**, který obsahuje definici všech karet ve hře a dané objekty i vytváří. Zde je tedy definováno, kdy karta může být zahrána, jaký má efekt na hru, kdy se daný efekt aplikuje, jaké má jméno nebo co se má stát po jejím zahrání.

Dalším důležitým souborem pro chod hry je **ScoreTable**. Jak už název napovídá, tak tento soubor má v sobě definici finálního hodnocení a na základě objektu hráče počítá jeho skóre.

Posledním důležitým souborem je **HumanPlayer**, který se od ostatních liší tím, že se už nejedná o věc potřebnou ke hře samotné. Jedná se o implementaci hráče, který předpokládá, že na něj bude přistupováno z jiného vlákna. Má v sobě naimplementovaný semafor a při zavolání metod hráče jen propaguje příslušná data na správné místo a čeká na odpověď. Na tuto implementaci je tedy napojeno grafické prostředí, které běží v druhém vláknu pro udržení interagování s programem, i když hráč není na tahu. Zároveň by na danou implementaci mohla být napojena webová stránka nebo jiný druh síťového připojení.

6.2 GameAI

V této složce tedy máme implementace všech použitých algoritmů. V případě, kdy by se měl přidat nový algoritmus, tak daná třída musí dědit od třídy **BaseAI**, která je ve stejné složce, a musí být zaregistrován v souboru **AIHolder**. Registrace probíhá jen tak, že se rozšíří enum **AINames** a ve funkci **GetAllAI** se přidá zavolání konstruktoru. To umožňuje implementovat konstruktor s vlastními parametry a zde je naplňovat.

Dále je tu složka **Data**, která obsahuje v současné době jen data pro MinMax algoritmus. Tedy obsahuje jak základní data, tak i data s naší strategií.

6.3 Extensions

V této složce jsou jak všechny rozšiřující funkce, tak i nové datové struktury, které se nedají zařadit do žádné jiné části. Tyto datové struktury nahrazují buď funkce, které nám kvůli starší verzi C# chybí, nebo užitečné typy, které dělají kód spekulativně čitelnější.

Datová struktura **Option** nahrazuje chybějící schopnost jazyka nullable references. Tato struktura umožňuje přenášet informaci o tom, zda nějaká hodnota může nabývat hodnoty null a tedy eliminuje nutnost kontroly proti null odkazu po vynoření z volaných funkcí. Tedy drží si buď danou hodnotu, nebo informaci o tom, že je prázdná. K tomu, aby tento koncept fungoval, tak se muselo zachovat dodržování několika podmínek napříč celým kódem. Hlavní podmínkou je, že žádná funkce nesmí vracet null, ale v takových případech musí vrátit Option.Empty.

Další datové struktury jsou jen pomocné pro čitelnost a přehlednost kódu. **Try** struktura například eliminuje rozepisování Try-Catch bloku a umožňuje řešení chyby

přesunout na vyšší úrovně. Poslední významnou strukturou je struktura **Cube**, která obaluje typ dvojitě zanořeného Dictionary. Cube zabaluje ověřování, zda daná kombinace klíčů už existuje a řešení co v daném případě udělat, a díky tomu je místo použití velmi přehledné.

Zbylé soubory obsahují rozšiřující funkce. Většina rozšiřujících funkcí jsou známé funkce, které jen místo chyby nebo null vrací Option. Jsou zde ovšem i funkce, které jen nahrazují běžně používané funkce a pojmenovávají je pro snadnější čtení. Mezi takové funkce patří například **IsNull**, která funguje stejně jako porovnání, zda proměnná má hodnotu null. Dále sem například patří funkce **ExceptNulls**, která vrátí danou kolekci bez všech výskytů hodnoty null. Podobných funkcí je zde mnoho, které slouží jen ke zvýšení čitelnosti a ulehčení vývoje, ale poslední, kterou bych zmínil, je funkce **Partition**, která rozdělí danou kolekci na dvě části podle poskytnuté podmínky.

6.4 Serialization

Je poslední velmi důležitou složkou potřebnou k vývoji, protože obsahuje všechny nástroje potřebné k práci se soubory na disku.

Hlavním souborem je zde **FileHandler**, který data, jak načítá, tak i ukládá. Při ukládání se mu dá parametrem funkce říct, zda má poskytnutá data převádět do formátu JSON či nikoliv. Jelikož data, která byla načítána, bylo vždy potřeba převézt z formátu JSON, tak možnost načíst data bez tohoto převodu, nebyla vytvořena.

Posledním důležitým souborem je zde **FileNameHandler**, který mimo jiné generuje z hry unikátní jméno ve formátu, který obsahuje už základní informace o hře. Používání daného formátu umožňuje rychle vyhodit data z načítané složky, které neobsahují data pro hru.

Ostatní soubory jsou pomocné funkce k předchozí funkcionalitě. Kromě funkcí k práci se soubory je zde i definován interface **ISerialize**, který slouží k označení struktur, které mohou být převedeny na primitivní typy a později přeformátovány do formátu JSON.

Závěr

Cílem práce bylo za pomocí umělé inteligence vytvořit obecnou strategii ke hře a ověřit, zda daná strategie funguje.

Výslednou strategii bych popsal jako úspěšnou. Algoritmus, který ji používá, je schopný vyrovnat se nebo porazit předchozí strategie a algoritmy i přesto, že hodnoty hodnotící funkce byly odhadnuty a jen trochu laděny. Výsledek bych tedy nepopsal jako návod, ale spíše jako dané milníky, ke kterým by měl hráč dojít. Danými milníky tedy jsou:

1. **Získání nového farmáře v období kolem první sklizně** a zároveň získání dostatečného množství jídla pro své farmáře jakýmkoliv způsobem.
2. **Začít vytvářet trvalý zdroj potravy** a obsazovat veškerá políčka farmy.
3. **Zajištění diverzity zvířat a surovin** ve střední části hry, tedy kolem 7. kola.
4. **Zisk nejvíce bodů v závěrečné části hry**, tedy někdy kolem 11. kola se tomu začít plně věnovat.

Při pokračování v ladění hodnot v hodnotící tabulce by šlo docílit lepšího výsledku, a tak zlepšit výslednou statistiku. Současné hodnoty byly vymyšleny mnou a jen trochu upravovány na základě krátkého pozorování. Pro optimální hodnocení by bylo potřeba zahrát tisíce dalších her a postupně upravovat skóre pro další hry a podle daných výsledků sledovat, kam se průměrné skóre posouvá.

Seznam tabulek

Tabulka 1: Výsledné ohodnocení hráče. 7
Tabulka 2: Porovnání výsledku algoritmů. 17
Tabulka 3: Porovnání výsledku algoritmů a navržené strategie..... 19
Tabulka 4: Porovnání hry Monte Carla, MinMax a naší strategie. 20

Seznam obrázků

Obrázek 1: Obsah deskové hry Agricola. (Zdroj https://mindok.cz/hra/agricola-zakladni-hra/) 3
Obrázek 2: Spuštění programu v příkazovém řádku. 21
Obrázek 3: Seznam všech příkazů a jejich parametrů s vysvětlením. 22
Obrázek 4: Ukázka výpisu informací při spuštění více her najednou. 23
Obrázek 5: Hlavní menu v grafickém okně. 24
Obrázek 6: Okno nové hry s výběrem množství protihráčů. 25
Obrázek 7: Okno z probíhající hry s označenými body zájmů. 25
Obrázek 8: Vyobrazení profesí v ruce hráče. 27
Obrázek 9: Menu pro načtení uložené nebo dokončené hry. 27
Obrázek 10: Okno s dokončenou hrou. 28
Obrázek 11: Menu pro výběr a spuštění hry umělých inteligencí. 29

Seznam použité literatury

[1] Pravidla [online]. [cit. 2023-05-10]. Uwe Rosenberg. Dostupné z: *http://www.mindok.cz/userfiles/files/pravidla/agricola_pravidla.pdf*

[2] WLASCHIN, Scott. Domain Modeling Made Functional*. P1.0. Raleigh, North Carolina: Pragmatic Bookshelf, 2018. ISBN 978-1-68050-254-1.*

[3] Data z her Agricola *[online]. [cit. 2023-05-10]. Dostupné také z:* *https://github.com/MichalVacek/AgricolaData*

[4] Monte Carlo tree search [online]. [cit. 2023-05-10]. Rahul Roy. Dostupné z: *https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts*

[5] Game Theory — The Minimax Algorithm Explained [online]. [cit. 2023-05-10]. Marissa Eppes. Dostupné z: *https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1*

[6] Alpha-Beta Pruning [online]. [cit. 2023-05-10]. JavaTpoint. Dostupné z: *https://www.javatpoint.com/ai-alpha-beta-pruning*

[7] Dokumentace Windows Forms [online]. [cit. 2023-05-10]. Microsoft. Dostupné z: *https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-7.0*

A Přílohy

A.1 Elektronická příloha

- Práce ve formátu PDF.
- Složka Raw obsahující složky CMD a GUI, ve kterých se nachází zkompilovaný program ve verzi příkazového řádku a grafického prostředí.
- Složka Solution s Visual Studio projektem.
- Složka Data, ve které je část uložených her. Konkrétně se jedná o hry algoritmu Monte Carlo.